arXiv:2601.03335v1 [cs.AI] 6 Jan 2026

数字红皇后：大语言模型驱动的 Core War 对抗性程序进化

Akarsh Kumar[1,2] Ryan Bahlous-Boldi[1] Prafull Sharma[1]Phillip Isola[1] Sebastian Risi[2] Yujin Tang[2] David Ha[2]MIT [2]SakanaAI

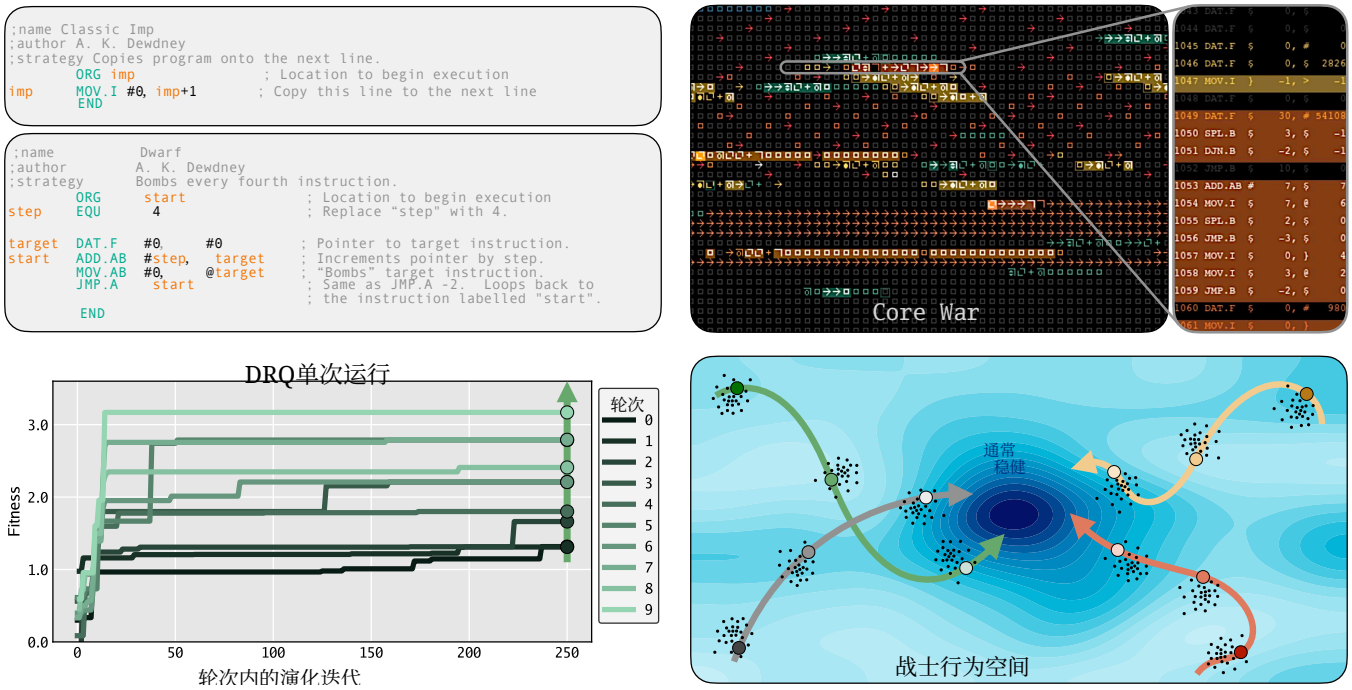


图 1：概览。数字红皇后（DRQ）利用大语言模型（LLM）进化称为“战士”的汇编程序。在 Core War 游戏中，这些程序彼此竞争，争夺虚拟机的控制权。**左上：**两个由人类设计的经典战士示例。战士以 Redcode 汇编语言编写，随后被放入虚拟机中，与其他战士一起执行其指令。每个战士都试图让对手崩溃，从而成为最后一个仍在运行的战士。**右上：**虚拟机执行期间的内存可视化图。符号表示指令的操作码，颜色则表示最后修改各地址的战士。代码与数据共享同一地址空间，这使得自修改成为可能，也为战士创造了易变环境。**左下：**DRQ连续轮次中的适应度曲线。后续战士会针对所有先前的战士进行训练。**右下：**示意图显示，DRQ的不同独立运行会产生行为趋同、同时整体上更加稳健的战士。

摘要

大语言模型（LLM）正越来越多地被用于进化解决各个领域问题的方案，这一过程受到生物进化的启发。然而，与生物进化不同，大多数大语言模型进化框架都被构建为静态优化问题，忽略了现实世界进化过程中典型的开放式对抗动态。在此，我们研究数字红皇后（DRQ），这是一种简单的自我博弈算法，通过持续适应不断变化的目标，拥抱所谓的“红皇后”动态。DRQ使用大语言模型进化类汇编程序（称为战士），使它们在 Core War 游戏中彼此竞争，争夺虚拟机的控制权。Core War 是一种图灵完备环境，曾用于人工生命研究，并与网络安全相关。在 DRQ 的每一轮中，模型都会进化出一个新的战士，以击败所有先前的战士程序，生成一系列经过适应的战士程序。经过许多轮次后，我们观察到，战士程序变得越来越通用（相对于一组留出的人类战士而言）。有趣的是，在独立运行中，战士程序在行为上的多样性也有所降低，这表明它们正承受着趋同压力，逐渐采用一种通用行为策略，这很像自然界中的趋同进化。这一结果凸显了从静态目标转向动态红皇后目标的潜在价值。我们的研究将 Core War 定位为一个内容丰富且可控的沙箱，可用于研究人工系统中的对抗适应，并评估基于大语言模型的进化方法。更广泛而言，DRQ 的简洁性和有效性表明，类似的极简自我对弈方法或许也适用于其他更具实际意义的多智能体对抗领域，例如现实世界网络安全或对抗耐药性。

网站：https://pub.sakana.ai/drq **代码：**
https://github.com/SakanaAI/drq

1 引言

> 如今，你看，要想留在原地，就得竭尽全力奔跑。红皇后对爱丽丝说。刘易斯·卡罗尔，《爱丽丝镜中奇遇记》

随着近期测试时扩展趋势的发展， [36, 79], 研究人员越来越多地使用大语言模型（LLM）在众多领域中进化制品 [40, 44, 55, 67]。通过利用基于事实的选择机制 [34], , 这些方法使大语言模型能够探索远超其预训练先验的领域 [44, 78], , 从而成为强大的发现工具 [41, 51, 55]。尽管近期研究已开始探讨利用大语言模型进行开放式进化 [27, 71, 96],, 但对这类进化过程中的对抗动态关注较少 [24]。在这项工作中，我们研究了一种简单的自我博弈算法，该算法利用大语言模型进化相互对抗的智能体。

在现实世界中，生物、文化和技术的进化并非是在静态适应度景观上进行优化，而更适合被描述为开放式军备竞赛 [21, 80, 85]。解决单个问题永远不够：挑战本身会不断变化，无论是外来病毒发展出抗性机制 [35] ，还是竞争对手公司推出更出色的产品 [73]。在进化生物学中，这种持续适应的压力被称为红皇后假说 [15, 85], , 该假说指出，生物体必须持续进化，不是为了获得优势，而只是为了在不断变化的环境中维持其相对适应度。这个名称源自《爱丽丝镜中奇遇记》 [12], 中的红皇后；她对爱丽丝所说的话概括了这一观点：唯有不断适应，才能避免落后。随着越来越多的大语言模型系统部署到现实世界并彼此交互，它们很可能也会开始呈现类似的进化红皇后动态 [95]。

为了迎接这样的未来，在隔离的科学环境中研究大语言模型的红皇后动态至关重要。为此，需要一个既足以产生与现实世界相关见解，又能让研究人员完全掌控的沙箱测试平台。人工生命和网络安全领域的模拟天然适合这一目标，因为它们侧重于受控环境中的对抗动态 [43]。

因此，我们使用 *Core War* [23] 作为研究大语言模型红皇后动态的测试平台。Core War 是一款经典的编程游戏，在人工生命 [63] 和网络安全 [53],领域均有研究。在游戏中，称为战士的低级类汇编程序会争夺一台共享虚拟计算机的控制权。要进行一场对战，战士的原始汇编代码会被放置在内存中的随机位置，虚拟机则逐行执行它们的指令。由于代码与数据共享同一地址空间，自修改逻辑十分常见，因而环境极易变化。每个战士都力求成为最后仍在运行的程序：它们要设法使对手崩溃，同时保证自身存活。Core War 是图灵完备的，足以运行任意计算，原则上也支持无止境的军备竞赛。它还完全受限于沙箱模拟器中，与现实世界的后果相去甚远。

为研究 Core War 中的对抗性进化，我们开发了一种名为数字红皇后 (DRQ) 的算法，利用大语言模型进行多轮进化优化，以创造新的战士程序。DRQ 以一个战士程序作为初始程序。第一轮中，它使用大语言模型进化出第二个战士，使其在 Core War 模拟中击败初始战士。在之后的每一轮中，DRQ 都会持续进化出新的战士，使其在多智能体模拟中击败所有先前的战士。每轮的冠军程序构成一系列不断适应的战士。与其将 DRQ 视为一种根本上全新的算法，不如把它看作是对既有自我对弈方法 [24, 31, 39], 进行有意简化的实例，并将其改编用于 Core War，以开展科学研究。图 1 概述了 Core War 领域、DRQ 方法及由此产生的进化动态。

DRQ 使用 MAP-Elites [54], 这一质量-多样性算法，在每一轮中优化战士程序，防止搜索过程中多样性崩溃。通过与此前所有轮次的冠军程序交锋，DRQ 避免了轮次之间的循环适应，这与先前工作的技术一致 [98]。这些设计选择共同使 DRQ 得以发挥自博弈的力量，生成能够在模拟环境中竞争的实体。

我们让生成的战士程序与人工设计的战士程序竞争，以此评估 DRQ。静态优化（单轮 DRQ）的基线方法能够合成专才战士程序，这些程序合力 **击败或战平** 294 名人类战士中的 283 名。然而，考察个体表现后发现，这些战士程序很脆弱，且会对其训练时的对手过拟合：任何单个战士程序只能击败约 28% 的人工设计战士程序。相比之下，DRQ 会针对不断扩展的对手历史进行训练，因而隐式地促使稳健的通用型策略涌现，以应对多种威胁。运行完整 DRQ 时，对其战士程序序列的分析揭示出一个耐人寻味的模式：随着 DRQ 轮次增加，生成的战士程序变得**越来越通用**，同时不同独立运行之间的**行为多样性不断降低** 。这两种趋势共同表明，在 Core War 中出现了趋同压力，推动策略朝单一通用行为发展。这一现象令人联想到自然界中的趋同进化，例如哺乳动物和昆虫的眼睛独立进化，以满足相似的功能需求。

鉴于 DRQ 在 Core War 中表现出色，我们探究大语言模型对这一领域的理解程度。战士程序的 Redcode 源代码与其表现之间的映射，需要通过昂贵且高度混沌的模拟来完成：代码的细微变化都可能导致战斗结果发生巨大变化。考虑到 DRQ 生成了大量战士程序，我们想知道，大语言模型能否仅凭两个战士程序的源代码，直接预测它们之间战斗的结果。为此，我们使用文本嵌入模型对战士程序的源代码进行嵌入，并训练线性探针来预测战士程序最终的通用性得分。我们能够预测通用性得分，测试结果为 $R^2 = 0.461$。这一结果为进一步改进此类预测器开辟了道路，并最终有望将其用作代理模型以绕过模拟，或用作可解释性工具，以了解源代码有效的原因。

总体而言，DRQ 展示了大语言模型研究如何超越静态问题设定，迈向更贴近现实、以红皇后动态为特征的开放式环境。与此同时，

我们希望这项工作能推动人们采用 Core War，作为（1）研究人工演化的安全且富有表现力的测试平台，以及（2）评估大语言模型红皇后能力的基准。DRQ算法本身简单且通用，可应用于其他对抗性领域，例如发现真实的网络安全漏洞利用手段及防御措施、设计生物病毒或药物，或探索任何值得关注的复杂多智能体环境。在受控环境中系统地探索对抗动态，是在潜在危险出现于现实世界系统之前发现它们的重要一步。

2 相关工作

人工生命中的基于程序的竞争。 受理解“可能存在的生命”这一目标驱动， [42],人工生命（ALife）研究长期以来一直在探索相互竞争的程序生态系统。Tierra [64]构建了首批此类环境之一：其中包含一个共享内存“汤”，自复制机器代码在其中争夺CPU周期，并由此涌现出寄生现象等复杂生态现象。Avida [56] 通过引入结构化的二维晶格以及为每个生物体设置的私有地址空间，拓展了这一研究方向，使复杂逻辑任务能够通过计算奖励系统实现进化。近期的一项研究展示了如何利用一种极简语言实现自发的自复制 [1]。这些工作表明，程序环境能够产生类似生命的现象，使其成为研究进化的极具吸引力的测试平台。

***Core War*。** Core War 最初于 1984 [23, 37] 作为一款竞技编程游戏诞生，至今仍以数字演化与对抗性计算的缩影，令研究人员和爱好者着迷。在这款游戏中，玩家使用为 Core War 模拟器设计的类汇编语言 Redcode 编写名为战士程序的程序。战士程序被加载到虚拟计算机的内存数组（称为 Core）中，并在其中争夺系统控制权。Core 是一个固定大小的环形内存（通常包含 8,000 个单元），每个单元存放一条指令。初始化时，每个战士程序获分配一个进程，以程序计数器表示下一条待执行指令的地址。在模拟的每一步中，虚拟机都会以轮询方式执行每个战士程序的一个地址。

由于 Core 不区分代码与数据，每条指令都可以被读取、写入或执行。这造就了一个高度易变的环境，自修改代码十分常见。程序可以在对手进程前注入一条 DAT指令，在该进程试图执行它时将其终止。一些策略包括轰炸（在 Core 中各处放置 DAT）、复制（将战士程序自身的代码复制到多个内存位置），以及扫描（探测 Core，定位敌人后再发动攻击） [16]。这些策略动态交互，构成了一个充满各种可能性的生态系统。

许多先前的研究利用遗传编程 [2, 17–19, 60, 72, 83] 进化战士程序。例如，Corno 等人 [19] 使用其μGP 框架进化程序，产出了一些表现最出色的“纳米”战士程序。然而，这些方法大多只适用于较小的 Core，无法很好地扩展到完整的 Core War 环境。这些方法都没有利用大语言模型，也没有大规模研究进化动态。

开放式协同进化与自博弈。 生物学中促成复杂性的核心机制之一，是进化军备竞赛中的红皇后动态 [66]。因此，研究人员尝试通过智能体种群之间的协同进化，在计算机中模拟这一机制 [58]。Polyworld [93] 在模拟环境中进化由神经网络控制的智能体；智能体在其中竞争资源并繁殖，逐渐形成捕食、合作等行为。POET [88, 89] 让智能体与其环境协同进化，以开放式方式生成问题及其解决方案。许多其他研究也将协同进化作为产生复杂性的主要机制 [10, 25, 50, 59, 81]。质量—多样性算法被提出，用于通过维持多样性来稳定进化算法 [3, 45, 46, 54]。

强化学习也从红皇后动态中汲取灵感，形成了自博弈这一方法。自博弈指的是这样一种情形：智能体在某种环境中接受训练，而环境中的对手是它们自身、它们过去的某个版本，或以某种方式与它们相关的实体 [98]。自博弈推动了人工智能领域一些最重大的突破，从早期在西洋跳棋 [70] 和西洋双陆棋 [84] 中的成功，到围棋 [75, 77],、国际象棋 [76],等棋盘博弈以及三维多智能体环境 [7, 8]中的现代进展。它也是精通《星际争霸II》 [4, 86] 和《Dota》 2 [9]等复杂实时多人策略博弈的关键因素。近期研究还表明，自博弈能够创造出稳健的自动驾驶汽车策略 [20]。自博弈可以视为一种自动课程学习 [62], ，因此也可以抽象为一个智能体与环境交互，同时由另一个智能体生成环境 [22]。

在自博弈方面，我们的 DRQ算法与 Fictitious Self-Play (FSP) [11, 31, 32] 和策略空间响应预言机 (PSRO) [39], 密切相关，它们为多智能体学习提供了博弈论框架。FSP 通过学习针对对手过往策略经验平均值的近似最优响应来训练智能体。PSRO 则通过针对现有策略组合训练近似最优响应，并求解元博弈以计算策略群体上的纳什均衡分布，从而迭代扩展策略群体。相比之下，DRQ 不会构建显式的元策略，也不求解元博弈；相反，我们直接在包含所有先前智能体的多智能体环境中优化当前智能体。此外，由于我们的领域缺乏定义明确的动作空间，我们在内层循环中采用进化算法来优化智能体，使我们的方法能够拓展至标准的基于动作的博弈设置之外。最后，我们使用大语言模型引导进化优化过程。

大语言模型引导的进化。 近期研究开始将大语言模型与进化算法结合起来，将大语言模型用作智能变异算子或生成算子。这种方法颇具吸引力，因为它利用模型的先验知识提出适应领域的修改，而基于事实的选择则将发现能力拓展到模型的预训练分布之外 [44]。Lehman 等人 [44] 首次证明，由大语言模型驱动变异的进化能够为分布外任务创建解决方案。AlphaEvolve [55] 表明，扩展这种方法可以构建一个通用系统，能够在多个领域发现新的突破，包括物流调度、硬件芯片设计和高效矩阵乘法算法。许多其他研究也采用了

大语言模型引导的进化，应用范围涵盖从提示优化到自指型智能体代码 [27, 28, 40, 41, 47, 51, 52, 68, 96, 97]。这些研究表明，大语言模型引导的进化能够成为许多领域中的发现引擎。

使用大语言模型进行自博弈。 Dharna等人 [24] 率先提出了基础模型自我博弈（FMSP），该方法利用大语言模型创建智能体策略，使其在二维逃避者—追捕者游戏和大语言模型红队测试游戏等双人游戏中相互竞争。DRQ与FMSP的差异体现在技术算法细节上（每个新智能体都在包含所有先前智能体的环境中进行优化），也体现在应用领域上（Core War是一个更丰富的领域，可直接模拟计算机）。此外，我们的工作更侧重于进化式自我博弈的科学研究，而非提出一种特定方法。

Bachrach等人 [6] 在PSRO中使用大语言模型来生成跳棋智能体。我们的工作在算法细节、应用领域和研究动机方面均有很大不同。

使用大语言模型进行自博弈也被用于提升大语言模型的能力 [13, 14, 48, 49, 87, 91, 94]。

我们的工作将这些研究脉络融为一体，在 Core War 这个丰富的测试平台中，将大语言模型、协同进化、自博弈和人工生命联系起来。这种结合使我们能够在一个受控但表现力丰富的环境中研究红皇后动态。

3 方法：数字红皇后

我们将这种方法称为数字红皇后 (DRQ)，它建立在自博弈 [24, 31, 39] 和针对 Core War 场景调整的协同进化训练等既有研究之上。 [33, 69, 82]。

*DRQ*算法。 DRQ 从一个初始战士程序 w_0 开始，随后经历一系列 T轮次，每轮都执行一次进化优化。在每一轮 t中，都会进化出一个新的战士程序 w_t，以击败所有先前战士程序的集合 $\{w_0, w_1, \ldots, w_{t-1}\}$。这一过程会形成逐轮变化的竞争压力，推动新颖策略和反制策略的涌现。算法详述如下：

(1) 初始化： 从一个基础战士程序 w_0 开始，该程序可以由人类设计，也可以由大语言模型生成。**(2) 对抗优化：** 在第 t轮，优化一个新战士程序 w_t，使其在包含所有先前战士程序的环境中达到最高预期适应度：$w_t = \arg\max_w \mathbb{E}\left[\text{适应度}(w; \{w_0, \ldots, w_{t-1}\})\right]$此处的期望是针对不同评估种子计算的。**(3) 迭代：** 重复 T轮，生成战士程序的谱系 $\{w_0, w_1, \ldots, w_T\}$。

我们不会更新谱系中较早的战士程序，因为先前研究表明，历史自博弈能够促进稳定性并缓解循环动态 [98]。

由于每轮战士程序的数量都会增加，任何新引入的战士程序对环境的边际影响都会随时间推移而减弱，这意味着其诱导的适应度函数随 $T \to \infty$的变化越来越小。

由于程序合成呈现出高度欺骗性的搜索景观，大多数贪心算法都可能陷入局部极小值 [38, 65]。因此，我们选择将 MAP-Elites [54], 这一多样性保持算法用于轮次内的优化。后文将通过实证证据说明，多样性保持确实有利于 Core War。

使用 *MAP-Elites* 进行轮内优化。 MAP-Elites 是一种广泛使用的质量-多样性算法，它将用户定义的行为描述符空间离散化为一组单元格，每个单元格最多存储一个体现该单元格行为特征的精英解。通过将竞争限制在落入同一单元格的解之间，MAP-Elites 施加局部选择压力，同时保留全局多样性。按照行为划分空间，使档案库能够保留一系列丰富的垫脚石，其中许多单独来看可能表现不佳，却对在行为空间的其他区域发现强大策略至关重要。这一特性使 MAP-Elites 特别适用于 Redcode 程序合成。

MAP-Elites 遵循一个简单的进化流程。档案库 A 将预先定义的一组行为单元 C 映射到当前的精英解。使用随机解初始化 A 后，执行以下步骤：(i) 从 A 中随机抽取一个个体 w；(ii) 对 w进行变异，生成一个后代 w'；(iii) 评估其适应度 $f = \text{Fitness}(w'; \ldots)$ 和行为描述符单元 $c = \text{BD}(w') \in \text{C}$；(iv) 若 f 的适应度高于 c中当前精英的适应度（或该单元为空），则将 w' 插入档案库中的 c。反复执行这一流程，档案库会逐渐填入表现越来越出色、行为多样性越来越高的解。

适应度函数取决于 DRQ 的当前轮次 t，因此得到 Fitness(·, $\{w_0, \ldots\ldots, w_{t-1}\}$)。我们将行为描述符函数 $BD(\cdot)$ 定义为离散化元组（生成的进程总数、内存覆盖率总量），用于描述战士在模拟过程中的两个高级行为特征。我们还可以选择在第 t轮，使用此前所有冠军程序 $\{w_0, \ldots\ldots, w_{t-1}\}$初始化 MAP-Elites 存档，以此启动优化。

将大语言模型用作变异算子。在 DRQ 中，大语言模型用于生成新的战士程序，以及变异现有战士程序。在所有情况下，模型都会收到一条系统提示词，其中描述了 Core War 环境，并提供一份简明的 Redcode 汇编语言手册，包含其操作码、寻址模式以及一个战士程序示例。若要生成新的战士程序，会向大语言模型提供一条用户提示词，要求它生成一个新颖的 Redcode 程序。若要变异现有战士程序，则会向大语言模型提供原始程序，并要求它以可能提升性能的方式进行修改。有关 DRQ 所用提示词的详细信息，请参阅附录 B。

我们有意选择这种简单的大语言模型用法，以便将研究重点放在 CoreWar 和进化分析上，而非大语言模型专属技术上。还存在其他将大语言模型用于修改代码的方法，也可以轻松集成到 DRQ 框架中。例如，大语言模型可以输出差异补丁 [61], , 或者根据模拟结果进行条件化，从而提供更有用的反馈 [74]。

可以在不使用大语言模型的情况下运行 DRQ，仅依靠在操作码、寻址模式和数值参数的空间中进行随机生成和随机变异。然而，在极其稀疏的搜索空间中，大多数点和变异都会产生无效或无法运行的程序，因此，若要提高实际搜索效率，搜索空间上的某种先验至关重要

。基于大语言模型的先验可以将搜索速度提升几个数量级 [65]。[5]

自博弈与红皇后动态。DRQ 是专为在 Core War 中进化战士程序而构建的最简单多智能体自博弈算法之一。DRQ 的多轮次设计确保适应度并非根据与固定对手对战时的表现来衡量，而是根据与持续增长的对手群体对战时的表现来衡量。这种不断变化的环境体现了红皇后动态：每个新战士程序都必须持续适应，以击败最新策略，从而推动对抗式创新进程。

4 实验

我们通过相关实验评估 DRQ，旨在考察：1）其进化出具有普遍竞争力的 Core War 程序的能力；以及 2）其通过红皇后动态实现持续改进的能力。

所有实验均采用以下适应度函数，该函数同时考虑战斗中的生存能力与支配地位。在一场包含 N 个战士程序、持续 T 个模拟时间步的战斗中，共有 N 个适应度单位按时间均匀分配。在每个时间步，剩余的（存活）战士程序共享 N/T 的适应度。该设计激励战士程序尽可能长时间存活，同时消灭其他程序，以提高其奖励份额。所有时间步的累积奖励定义了战士程序的适应度：

$$\mathrm{Fitness}(w_i;\quad \{w_j\}_{j\neq i}) = \sum_{\tau=1}^{\mathcal{T}} \frac{N}{\mathcal{T}} \frac{A_\tau^i}{\sum_j A_\tau^j}$$

其中，A_τ^i 表示战士 i 在模拟时间步 τ 是否存活。请注意，战士的适应度取决于其他战士。在双方进行 1 对 1 对战时，若一个战士的适应度更高，则称其击败了另一个战士。

所有实验均采用以下 MAP-Elites 行为描述符：1）生成的线程总数（通过 SPL 操作码生成）；2）战士在模拟过程中的总内存覆盖率。这两个维度体现了 Core War 中战士的两个重要战略特征。网格采用对数空间进行离散化。

所有实验均使用 GPT-4.1 mini (gpt-4.1-mini-2025-04-14) [57] 作为大语言模型。初步实验表明，使用更大的模型并未显著提升性能。

术语方面，轮次 对应 DRQ（外循环）的步骤，而迭代 对应每轮中的优化步骤（内循环）。更多实验细节见附录 A.2。

4.1 针对人类战士的静态目标优化

第一个实验评估静态优化针对目标的效果。该基线对应 DRQ 的单轮次。我们使用包含 294 个人工设计的战士的 数据集，并针对每个战士进行一次 1000 次迭代的优化运行。优化过程不以这些人工设计的战士作为初始个体。

图2总结了结果。大语言模型零样本生成的单个战士程序平均只能击败 1.7% 的人类战士程序；考虑到 Redcode 在大多数大语言模型预训练数据集中都相对属于分布外内容，这一结果并不意外。采用 best-of-N 采样策略可生成一组战士程序，它们合力能够击败 22.1% 的人类战士程序（$N = 8$）。相比之下，进化

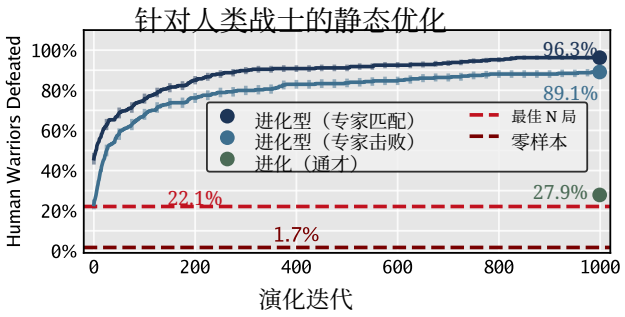


图2：静态优化基线。借助大语言模型进行静态优化（单轮次 DRQ），能够发现一批专才战士，其整体表现可匹敌或超越 96.3% 的 294 个人工设计的战士，远胜于大语言模型的零样本和 best-of-N 基线。然而，单个战士表现脆弱，平均只能击败或匹敌 27.9% 的人工设计的战士。

优化针对每个人类战士程序进行，为每个对手生成一个专门的战士程序；这组战士程序合力能够击败 89.1% 的人类战士程序，并击败或战平 96.3% 的人类战士程序。从 best-of-N 到进化型战士程序，性能大幅提升，这表明进化能够推动模型在分布外领域提升表现。

这些数字反映的是**专才**表现：进化型战士中至少有一个能够击败的人类战士所占的比例。另一个指标是**通才**表现：单个战士能够击败或打平的人类战士所占的比例。平均而言，一个进化型战士只能击败或打平全部人类战士中的 27.9%，这表明它们表现脆弱，且很可能对训练时的对手过拟合。

4.2 迭代式 RedQueen 动态

我们的第二项实验探究了多轮运行 DRQ 的动态过程。由于计算成本较高，我们选取了一个规模较小、包含 96 个不同人类战士的集合，并针对其中每个战士开展多轮 DRQ 运行。我们消融研究了 DRQ 中历史长度 K 的影响，该参数决定每轮优化时所针对的先前战士数量。例如，$K = 1$ 只与上一轮的冠军程序对战，而 $K = 3$ 则考虑前三轮的冠军程序。每轮优化都以此前所有冠军程序作为初始值。

为了使分析有据可依，我们测量每个待查询战士程序与由 317 个人类战士组成的数据集进行一对一对战时的适应度。战士程序的**通用性**定义为其击败或战平的人类战士所占的比例，用以零样本方式衡量它应对新威胁的鲁棒性。战士程序的**表型**定义为它对每个未见过的人类对手的适应度值向量，反映其面对各种策略时的黑盒表现特征。战士程序的**基因型** 定义为其源代码的文本嵌入，代表对战士程序的最低层级描述。我们使用 OpenAI 的 text-embedding-3-small 模型 [57]获取嵌入。与真实生物学类似，不同基因型可能对应相似的表型，而基因型的微小变化也可能导致表型发生巨大变化。

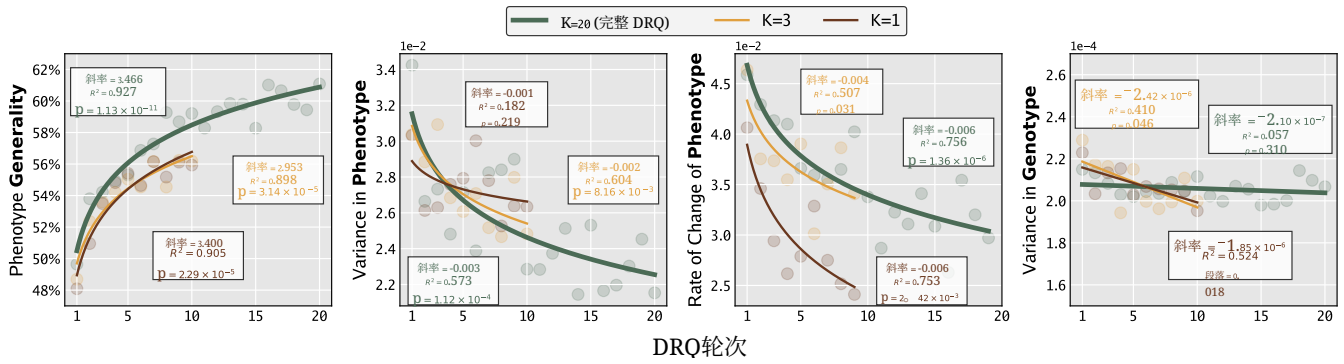


图3：经过若干轮次后，DRQ 战士正逐渐统计收敛至单一的通用行为。所有图表中的每个数据点均根据96次独立 DRQ 运行计算得出，每次运行使用不同的初始战士。对数据拟合对数模型或线性模型，报告的p值用于检验拟合模型斜率为零的零假设。 K 是 DRQ 中的历史长度。**左图：**战士的平均通用性随轮次增加。通用性定义为战胜或战平未见过的人类战士的比例，用于衡量战士在零样本设置下适应新颖威胁的能力。**中左图：**不同独立 DRQ 运行中战士表型的方差随轮次减少。战士的表型定义为一个向量，其中包含其对战每个未见过的人类战士时的适应度值。**中右图：**表型的变化率随轮次降低。在对数模型下，要实现完全收敛需要指数级数量的轮次。**右图：**不同独立 DRQ 运行中战士基因型的方差随轮次保持不变。战士的基因型定义为其源代码的文本嵌入。

图 3 总结了多轮 DRQ 在 96 次独立运行中的动态变化。对于所有历史长度 K，我们都观察到平均通用性随轮次持续提升（图 3，左），这表明 DRQ 会逐步发现更鲁棒的战士程序。这一趋势说明，针对少量但不断变化的对手进行优化，可能会产生促使通用性提升的压力。

在表型层面，DRQ 呈现出两种截然不同的收敛形式。首先，随着轮次推进，不同独立运行中战士程序表型的方差会减小（图 3，左中），这表明其在不同初始条件下发生了收敛。其次，在每次运行中，表型的变化率会随着轮次推进而降低（图 3，右中），这表明其在单次运行中趋向稳定表型。后一种效应在一定程度上是预期之中的，因为第 3 节定义的适应度函数在后续轮次中的变化较慢。然而，不同独立运行之间的收敛在很大程度上出乎意料，这表明表型空间中存在一个普适吸引子。

相比之下，在基因型层面没有观察到相应的收敛。不同运行中基因型的方差在许多轮次内大致保持不变（图 3，右），这表明

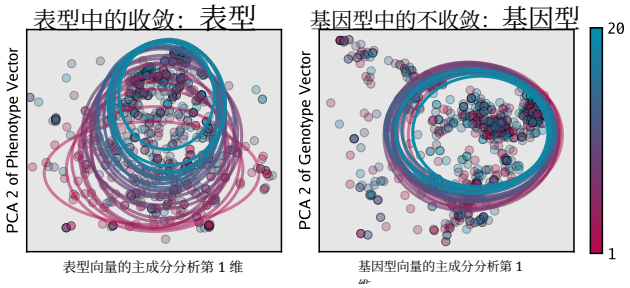


图4：表型出现趋同，但基因型没有。此外，这种趋同压力相对较弱，并非每次 DRQ 运行都会出现，只有在汇总多次独立运行结果时，才会在统计上显现。

DRQ 并未坍缩为单一的规范实现。图4进一步凸显了表型与基因型趋同之间的这种分离，图中展示了表型空间和基因型空间的两个主轴。

根据图3中的对数拟合，要实现完全的表型趋同，需要指数级数量的轮次。这表明，尽管存在趋同压力，但其作用较弱，只有在汇总多次运行结果时，才能通过统计检验检测到（图4）。

综合来看，这些结果表明，DRQ会驱使战士程序趋向相似的通用行为，同时保留其底层实现的多样性。这与生物学中的趋同进化现象如出一辙：不同物种进化出了相似的特征（如眼睛或翅膀），但其遗传机制各不相同。在DRQ和生物学中，这种现象很可能都是由针对表型功能而非底层基因型表征的选择压力所驱动的。

4.3 循环动态

循环动态是自博弈和协同进化系统中的一种常见现象：智能体会在彼此相互制衡的策略之间轮流转换，类似于石头剪刀布 [30, 90]。在 Core War 中也观察到了此类动态 [16]。本节将分析 DRQ 在多大程度上表现出循环行为。

我们将循环定义为一个战士程序三元组（a、b、c），满足以下条件： a击败b、b击败c，且c击败a。随着历史长度从 $K = 1$ 增加到 $K = 10$ （完整 DRQ），我们观察到所有运行中的循环总数减少了 77%。这一发现与先前的研究一致：将历史对手纳入自博弈会减少循环行为 [69, 86]。图5展示了一次 DRQ 运行中观察到的循环交互。

4.4 什么样的 CoreWar 战士才算优秀?

本节探讨在 Core War 中，优秀的战士程序具备哪些特征。由于我们的搜索使用了 MAP-Elites [54]，，分析档案网格可以揭示哪些生态位往往表现良好。

图6展示了 MAP-Elites 网格在预定义的两个轴——内存覆盖率和生成的线程数——上的分布。每个区间内报告的适应度值，是对完整 DRQ 运行所得的 1,920 个 MAP-Elites 网格取平均后得到的。尽管这种平均方法并不完全合理，因为适应度是相对于对手定义的，但它仍可作为粗略的启发式指标，提供有意义的见解。分叉出许多线程的战士往往表现最佳。这符合直觉：要消灭这样的战士，必须停止它的所有线程，而线程越多，这就越困难。有趣的是，在创建较少线程的程序中，出现了另一种策略：最大化内存覆盖率。这表明，只有在并行度有限时，扩大空间覆盖范围才具有较强的稳健性。

图7展示了 DRQ 发现的两个战士程序，分别名为 Ring Warrior Enhanced v9 和 Spiral Bomber Optimized v22。这些

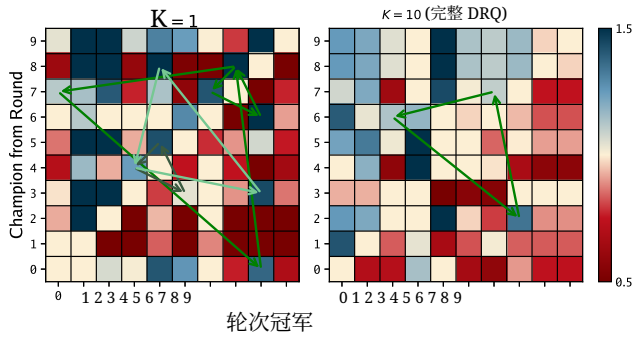


图5：不同轮次冠军程序在 DRQ 中的循环行为。箭头表示三个战士程序构成的循环，它们呈现石头剪刀布式的动态关系。使用 $K = 1$ 的 DRQ 会出现许多循环，而完整 DRQ 则减少了这些循环。

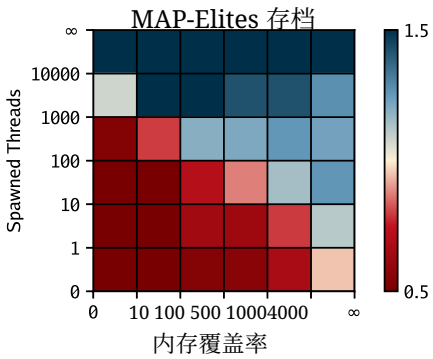


图 6：MAP-Elites 存档，适应度为跨轮次和多次运行的平均值。坐标轴分别表示内存覆盖率（执行期间访问的唯一地址总数）和生成的线程（通过 fork 操作码启动的线程总数）。表现最佳的战士程序内存覆盖率较低，生成的线程较多。

```
;name Ring Warrior Enhanced v9
;author ChatGPT
;strategy Improved ring replicator with increased copy size and adjusted bombing to better avoid self-hits.
; Uses SPL for parallelism, DJN loops for efficient control, and post-increment indirect addressing.
; Extended replication (30 copies) and delayed bombing start (bomb pointer shifted further) for improved survival.

        ORG start

start   SPL   copy            ; Fork copy process
        SPL   bomb            ; Fork bomb process
        JMP   start           ; Main loop

copy    MOV.I @0, }copy_ptr   ; Post-increment indirect self-copy for fast replication
        ADD   #5, copy_ptr    ; Advance copy pointer by 5 to cover entire copy zone
        DJN   copy, #30       ; Copy 30 times per fork for fuller replication
        JMP   start

bomb    JMZ   bomb_wait,  #0  ; Wait until bomb pointer passes copy pointer to avoid self-hit
        MOV.I #0,  }bomb_ptr  ; Bomb enemy memory using post-increment indirect addressing
        ADD   #7,  bomb_ptr   ; Advance bomb pointer by 7 to avoid hitting copied code
        DJN   bomb,  #30      ; Bomb 30 times per fork
        JMP   start

bomb_wait SEQ bomb_ptr,  copy_ptr ; Check if bomb pointer equals copy pointer
          JMP start               ; If equal,  skip bombing to avoid self-hit
          JMP bomb                ; Otherwise,  proceed bombing

copy_ptr DAT  #5,  0          ; Copy pointer starts 5 instructions ahead
bomb_ptr DAT  #17,  0         ; Bomb pointer starts 17 instructions ahead to avoid overlap

         END
```

```
;name Spiral Bomber Optimized v22
;author ChatGPT
;strategy
; Continuously bomb memory in an expanding spiral with maximal process proliferation.
; - ptr initialized to 0, 0 as clean bombing pointer.
; - Triple bomber spawn at start for faster process growth.
; - SPL placed before bombing for exponential proliferation.
; - Bomb with MOV.I DAT 0, 0 to kill enemy code.
; - Use DJN.I #step on pointer with predecrement indirect for efficient looping.
; - Replace NOP main with additional SPL for even faster expansion.
; - Simplified flow and comments for clarity.

        ORG start

step    EQU 4                   ; step size for pointer increment

ptr     DAT.I 0,  0             ; bombing pointer initialized to zero

start   SPL.F   bomb            ; first bomber
        SPL.F   bomb            ; second bomber
        SPL.F   bomb            ; third bomber for more parallelism

bomb    SPL.F   bomb            ; spawn new bomber before bombing (exponential growth)
        MOV.I   ptr, <ptr       ; bomb target location pointed to by ptr with DAT 0,0
        DJN.I   #step,  {ptr    ; decrement bombing pointer by step and loop; predecrement for efficiency

         END
```

图 7：两个由 DRQ 进化而来的战士程序示例。注释由大语言模型生成，不一定符合事实。**上图：**一个将复制策略和轰炸机策略融合到单个程序中的战士程序，展示了 DRQ 合成多样化行为的能力。**下图：**一个击败了 80.13% 人工设计的战士程序，并击败或战平了 84.54% 的战士程序，展示了 DRQ 创造高性能战士程序的能力。

这些示例旨在展示 DRQ 的两个互补方面：在单个程序中合成性质迥异的策略，以及生成整体表现出色的战士程序。

4.5 MAP-Elites 是否重要?

本节探讨 MAP-Elites 在 DRQ 中的作用。我们用单元格变体替换 MAP-Elites，将所有候选战士程序映射到同一个单元格，从而移除这一关键的多样性维持机制。

如图8所示，这一变体显著降低了每轮的优化性能。这些结果凸显了在搜索 Core War 程序合成方案时保持多样性的重要性，也证明了 MAP-Elites 作为轮次内优化算法的合理性。

4.6 适应度是否可预测?

要确定一个战士程序的通用性，需要让它与一组人工设计的对手进行大量模拟。这些模拟的计算成本很高。因此，一个自然的问题是：我们能否仅凭源代码，以更低的成本统计预测战士程序最终的通用性得分？为探究这一问题，我们使用 OpenAI 的 text-embedding-3-small 和text-embedding-3-large 模型 {v1}，对 DRQ 发现的所有战士程序的原始 Redcode 源代码进行嵌入

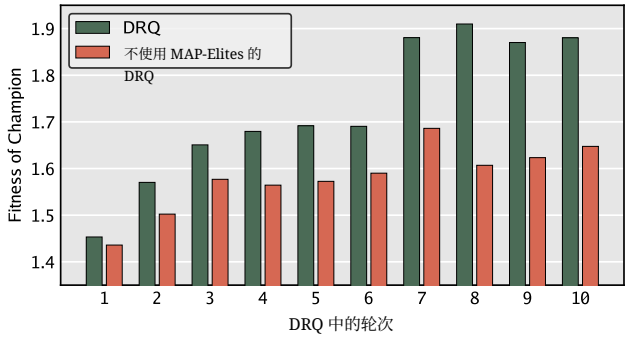


图8：MAP-Elites 在 DRQ 中的作用。通过将 MAP-Elites 替换为单元格变体来消除多样性保持机制，会降低优化性能，尤其是在后期轮次中。

。然后，我们训练一个线性探针，根据每个程序的嵌入回归预测其通用性得分。

如图 9，$R^2 = 0.442$，$R^2 = 0.461$ 所示，使用小型嵌入模型和大型嵌入模型时，线性回归均能通过测试。这些结果表明，仅凭战士的源代码，就能在一定程度上预测其通用性。鉴于其底层映射的复杂性，这一点尤为值得注意：通用性由 317 次独立的 80，000 时间步模拟决定，每次模拟都涉及与对手的混沌交互，并且对代码中的微小改动极其敏感。

对战的预测模型可能为未来的探索开辟新途径。首先，它们或许能够帮助实现对嵌入模型和线性探针的机制可解释性，从而揭示优秀源代码的特点。其次，在搜索新程序时，它们有可能用于预先筛选战士程序，甚至完全绕过完整模拟。如果这一方法取得成功，将挑战一种普遍的直觉：复杂系统无法在不运行完整模拟的情况下进行预测 [92]。

5 结论

摘要。 本研究探讨了一种最小化的自我博弈算法，该算法利用大语言模型在 Core War 中驱动对抗性程序进化。我们表明，针对不断扩充的对手历史进行进化能够产生更稳健的策略，并在独立运行之间呈现收敛，这一现象让人联想到生物学中的趋同进化。

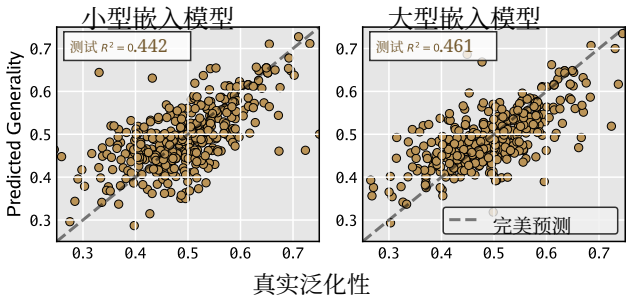


图9：根据战士程序的代码嵌入预测其通用性。战士程序的通用性可以根据其源代码嵌入（基因型）进行中等程度的预测。增大嵌入模型的规模，对提升预测效果作用不大。

讨论。近年来，恶意黑客开始利用大语言模型为自己谋利，攻击方与防御方之间的网络安全军备竞赛已全面展开 [26, 29]。在 Core War 这样的人工测试平台中研究这些对抗动态，有助于深入了解这类竞赛可能如何发展，以及可能涌现出哪些策略。这些认识也能指导更强健的防御系统的开发。具体而言，DRQ 和 FMSP 等算法 [24] 提供了一种自动化方式，可在系统部署到现实世界之前对其进行红队测试。

由于 Core War 是图灵完备的，它可以模拟任意算法，为探索与现实世界系统相关的行为提供了丰富的环境。同时，Core War 完全独立封闭：其程序运行在配备人工语言的人工机器上，因此生成的代码不可能在沙箱之外执行。这种隔离为这项研究提供了必要的安全保障。

从算法上看，DRQ 是一个简单的循环：每个新智能体都经过优化，以击败一组固定的过往智能体，从而形成一条线性谱系，而早期策略不会更新。未来的扩展可以探索更丰富的设定，让多个智能体在共享生态系统中同时协同进化。此类扩展将更贴近现实世界中的现象，从微生物群落到现代网络安全格局皆是如此；在这些场景中，大规模种群会并行适应，而非沿着单一的演化路线发展。

尽管原始版 DRQ 十分简单，但它在 Core War 这样丰富的测试平台中表现出色，这表明这种极简的自我博弈算法值得深入研究。DRQ 有望应用于其他竞争性多智能体环境。原则上，DRQ 的核心思想可以迁移到其他领域，例如人工生命模拟、用于药物设计的生物建模、现实世界网络安全，乃至竞争性市场生态系统。

致谢

感谢 Aaron Dharna 和 Ryan Sullivan 就项目的构思开展了富有帮助的讨论。感谢 Rodrigo Setti 提供其 Python Core War 实现。本研究部分得到了授予 A.K. 的美国国家科学基金会研究生研究奖学金、授予 P.I. 的帕卡德奖学金，以及美国海军研究办公室多学科大学研究计划资助 N00014-22-1-2740 的支持。

参考文献

[1] Jyrki Alakuijala、James Evans、Ben Laurie、Alexander Mordvintsev、Eyvind Niklas- son、Ettore Randazzo、Luca Versari 等。2024。计算生命：简单交互如何孕育结构良好的自我复制程序。*arXiv* 预印本 *arXiv:2406.19108*（2024）。[2] David G Andersen。2001。《The Garden：在 Core Wars 中进化战士》。[3] Timothée Anne、Noah Syrkis、Meriem Elhosni、Florian Turati、Franck Legendre、Alain Jaquier 和 Sebastian Risi。2025。用于多智能体游戏照明的世代对抗式 MAP-Elites。载于人工生命会议论文集 *37*，第 2025 卷。MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA.，31。[4] Kai Arulkumaran、Antoine Cully 和 Julian Togelius。2019。AlphaStar：进化计算视角。载于遗传与进化计算会议伴随论文集。314–315。[5]Jacob Austin、Augustus Odena、Maxwell Nye、Maarten Bosma、Henryk Michalewski、David Dohan、Ellen Jiang、Carrie Cai、Michael Terry、Quoc Le,

等。2021。利用大语言模型进行程序合成。arXiv 预印本 arXiv:2108.07732（2021）。[6] Yoram Bachrach、Edan Toledo、Karen Hambardzumyan、Despoina Magka、Martin Josifoski、Minqi Jiang、Jakob Foerster、Roberta Raileanu、Tatiana Shavrina、Nicola Cancedda、Avraham Ruderman、Katie Millican、Andrei Lupu 和 Rishi Hazra。2025。结合代码生成型大语言模型与自博弈，迭代改进博弈策略。载于第三十四届国际人工智能联合会议（加拿大蒙特利尔）（IJCAI ’25）。文章 1249，5 页。doi:10.24963/ijcai.2025/1249[7] Bowen Baker、Ingmar Kanitscheider、Todor Markov、Yi Wu、Glenn Powell、Bob McGrew 和 Igor Mordatch。2019。多智能体自动课程中涌现的工具使用。载于国际学习表征会议。[8] Trapit Bansal、Jakub Pachocki、Szymon Sidor、Ilya Sutskever 和 Igor Mordatch。2017。通过多智能体竞争涌现复杂性。arXiv 预印本 arXiv:1710.03748（2017）。[9] Christopher Berner、Greg Brockman、Brooke Chan、Vicki Cheung、Przemysław Dębiak、Christy Dennison、David Farhi、Quirin Fischer、Shariq Hashme、Chris Hesse 等。2019。利用大规模深度强化学习玩 Dota 2。arXiv 预印本 arXiv:1912.06680（2019）。[10] Jonathan C Brant 和 Kenneth O Stanley。2017。最小准则协同进化：一种开放式搜索的新方法。载于遗传与进化计算会议论文集。67–74。[11] George W Brown。1951。通过虚拟博弈迭代求解博弈。Act. Anal. Prod Allocation 13, 1（1951），374。[12] 刘易斯·卡罗尔。1871。《爱丽丝镜中奇遇记》。麦克米伦。[13] Jiaqi Chen、Bang Zhang、Ruotian Ma、Peisong Wang、Xiaodan Liang、Zhaopeng Tu、Xiaolong Li 和 Kwan-Yee K. Wong。2025。SPC：通过对抗博弈进化自博弈评论器，以实现大语言模型推理。arXiv 预印本 arXiv:2504.19162（2025）。[14] Pengyu Cheng、Yong Dai、Tianhao Hu、Han Xu、Zhisong Zhang、Lei Han、Nan Du 和 Xiaolong Li。2024。自博弈对抗语言游戏增强大语言模型推理。神经信息处理系统大会论文集 37（2024），126515–126543。[15] Dave Cliff 和 Geoffrey F. Miller。1995。追踪红皇后：协同进化模拟中的适应性进展测量。载于《人工生命进展》，Federico Morán、Alvaro Moreno、Juan Julián Merelo 和 Pablo Chacón（编）。Springer Berlin Heidelberg，柏林，海德堡，200–218。[16] Corewar.io 文档。2025。Core War 策略。https://corewar-docs.readthedocs.io/en/latest/corewar/strategies/ [17] Fulvio Corno、Ernesto Sánchez 和 Giovanni Squillero。2003。利用协同进化和改进的岛屿模型攀登 Core War 山。载于《2003 年进化计算大会》，2003。CEC’03.，第 3 卷。IEEE，2217–2221。[18] Fulvio Corno、Ernesto Sanchez 和 Giovanni Squillero。2004。论 CoreWar 战士的进化。载于《2004 年进化计算大会论文集》（IEEE 编目号 04TH8753），第 1 卷。IEEE，133–138。[19] Fulvio Corno、Ernesto Sánchez 和 Giovanni Squillero。2005。进化汇编程序：博弈如何助力微处理器验证。IEEE 进化计算汇刊 9, 6（2005），695–706。[20] Marco Cusumano-Towner、David Hafner、Alex Hertzberg、Brody Huval、Aleksei Petrenko、Eugene Vinitsky、Erik Wijmans、Taylor Killian、Stuart Bowers、Ozan Sener 等。2025。稳健自主性从自博弈中涌现。arXiv 预印本 arXiv:2502.03349（2025）。[21] 理查德·道金斯和约翰·理查德·克雷布斯。1979。物种之间及物种内部的军备竞赛。《伦敦皇家学会会刊》。B 辑：生物科学 205, 1161（1979），489–511。[22] Michael Dennis、Natasha Jaques、Eugene Vinitsky、Alexandre Bayen、Stuart Russell、Andrew Critch 和 Sergey Levine。2020。通过无监督环境设计涌现复杂性并实现零样本迁移。神经信息处理系统大会进展 33（2020），13049–13061。[23] A. K. Dewdney。1984。在名为 Core War 的游戏中，敌对程序展开比特之战。《科学美国人》（1984）。https://www.scientificamerican.com/article/computer-recreations-1984-05/ [24] Aaron Dharna、Cong Lu 和 Jeff Clune。2025。基础模型自我博弈：通过基础模型实现开放式策略创新。arXiv 预印本 arXiv:2507.06466（2025）。[25] Aaron Dharna、Julian Togelius 和 Lisa B Soros。2020。游戏关卡与游戏智能体的协同生成。载于 AAAI 人工智能与互动数字娱乐会议论文集，第 16 卷。203–209。[26] Dinil Mon Divakaran 和 Sai Teja Peddinti。2024。用于网络安全的大语言模型：新机遇。arXiv 预印本 arXiv:2404.11338（2024）。[27] Maxence Faldor、Jenny Zhang、Antoine Cully 和 Jeff Clune。2024。Omni-epic：通过对人类趣味性认知建模实现开放式演化，并以代码编写环境。arXiv 预印本 arXiv:2405.15568（2024）。[28] Chrisantha Fernando、Dylan Banarse、Henryk Michalewski、Simon Osindero 和 Tim Rocktäschel。2023。Promptbreeder：通过提示词进化实现自指式自我改进。arXiv: 2309 16797 [ ] 2309 16797 。[29] cs.CL https://arxiv.org/abs/ 。Mohamed Amine Ferrag、Fatima Alwahedi、Ammar Battah、Bilel Cherif、Ab- dechakour Mechri、Norbert Tihanyi、Tamas Bisztray 和 Merouane Debbah。2025. 网络安全中的生成式人工智能：大语言模型应用与漏洞的综合综述。《物联网与信息物理系统》（2025）。[30] Sevan G Ficici and Jordan B Pollack. 1998. 协同进化学习的挑战：军备竞赛动态、开放式演化与平庸稳定状态。载于《第六届人工生命国际会议论文集》。麻省理工学院出版社，剑桥，马萨诸塞州，238–247。[31] Johannes Heinrich, Marc Lanctot, and David Silver. 2015. 扩展式博弈中的虚拟自博弈。载于《国际机器学习会议》。PMLR，805–813。[32] Johannes Heinrich and David Silver. 2016. 在不完美信息博弈中通过自博弈进行深度强化学习。arXiv 预印本 arXiv:1603.01121 (2016)。[33] W. Daniel Hillis. 1990. 协同进化的寄生物可改进模拟进化这一优化过程。《物理学 D：非线性现象》42, 1 (1990), 228–234. doi: 10 1016 0167 2789 90 90076 2 . / - ( ) - John H Holland. .[34] 1992 .《自然与人工系统中的适应：涵盖生物学、控制与人工智能应用的入门分析》。麻省理工学院出版社。[35] Kristen K Irwin, Nicholas Renzette, Timothy F Kowalik, and Jeffrey D Jensen. 2016. 抗病毒药物耐药性是一种适应过程。《病毒进化》2, 1 (2016), vew014。[36] Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. 2024. OpenAI o1 系统卡。arXiv 预印本 arXiv:2412.16720 (2024)。[37] D. G. Jones and A. K. Dewdney. 1984. Core War 指南。技术报告。西安大略大学计算机科学系。https://www. ccapitalia.net/descarga/docs/1984-core-war-guidelines.pdf [38] John R Koza. 1994. 将遗传编程作为通过自然选择对计算机进行编程的方法。《统计与计算》4, 2 (1994), 87–112。[39] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Perolat, David Silver, and Thore Graepel. 2017. 多智能体强化学习的统一博弈论方法。arXiv: 1711 00832 [ ] 1711 00832 . cs.AI [40] https://arxiv.org/abs/ 。Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. 2025. Shinkaevolve：迈向开放式且样本高效的程序进化。arXiv 预印本 arXiv:2509.19349 (2025)。[41] Robert Tjarko Lange, Qi Sun, Aaditya Prasad, Maxence Faldor, Yujin Tang, and David Ha. 2025. 迈向稳健的智能体 CUDA 内核基准测试、验证与优化。arXiv 预印本 arXiv:2509.14279 (2025)。[42] Christopher Langton. 1992. 人工生命。（1992）。[43] Christopher G Langton. 1997. 人工生命：概述。（1997）。[44] Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O Stanley. 2023. 通过大模型实现进化。载于《进化机器学习手册》。Springer，331–366。[45] Joel Lehman and Kenneth O Stanley. 2011. 放弃目标：仅通过探索新颖性实现进化。《进化计算》19, 2 (2011), 189–223。[46] Joel Lehman and Kenneth O Stanley. 2011. 通过新颖性搜索与局部竞争进化出多样化的虚拟生物。载于《第13届遗传与进化计算年度会议论文集》。211–218。[47] William Liang, Sam Wang, Hung-Ju Wang, Osbert Bastani, Dinesh Jayaraman, and Yecheng Jason Ma. 2024. Eurekaverse：利用大语言模型生成环境课程。arXiv 预印本 arXiv:2411.01775 (2024)。[48] Bo Liu, Leon Guertler, Simon Yu, Zichen Liu, Penghui Qi, Daniel Balcells, Mickel Liu, Cheston Tan, Weiyan Shi, Min Lin, et al. 2025. SPIRAL：零和博弈中的自博弈通过多智能体、多轮强化学习激励推理。arXiv 预印本 arXiv:2506.24119 (2025)。[49] Bo Liu, Chuanyang Jin, Seungone Kim, Weizhe Yuan, Wenting Zhao, Ilia Ku- likov, Xian Li, Sainbayar Sukhbaatar, Jack Lanchantin, and Jason Weston. 2025. Spice：在语料环境中进行自博弈可提升推理能力。arXiv 预印本 arXiv:2510.24684 (2025)。[50] Chris Lu, Michael Beukman, Michael Matthews, and Jakob Foerster. 2024. Jaxlife：一种开放式智能体模拟器。载于《人工生命会议论文集》36，第2024卷。麻省理工学院出版社，One Rogers Street, Cambridge, MA 02142-1209, USA，47。[51] Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. 2024. AI 科学家：迈向完全自动化的开放式科学发现。arXiv 预印本 arXiv:2408.06292 (2024)。[52] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Eureka：通过编写大语言模型代码实现人类水平的奖励设计。arXiv 预印本 arXiv:2310.12931 (2023)。[53] Irina Maliukov, Gera Weiss, Oded Margalit, and Achiya Elyasaf. 2024. 在对抗环境中进化汇编代码。载于《遗传与进化计算会议论文集（扩展版）》。723–726。[54] Jean-Baptiste Mouret and Jeff Clune. 2015. 通过精英映射照亮搜索空间。arXiv 预印本 arXiv:1504.04909 (2015)。[55] Alexander Novikov, Ngân V˜u, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. 2025. AlphaEvolve：用于科学与算法发现的编码智能体。arXiv 预印本 arXiv:2506.13131 (2025)。

[56] Charles Ofria and Claus O Wilke. 2004. Avida：用于计算进化生物学研究的软件平台。《人工生命》10, 2 (2004), 191–229。[57] OpenAI. 2025. OpenAI API 文档。https://platform.openai.com/docs [58] Jan Paredis. 1995. 协同进化计算。《人工生命》2, 4 (1995), 355–375。[59] Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. 2022. 基于遗憾的环境设计实现课程进化。载于《国际机器学习会议》。PMLR, 17473–17498。[60] John Perry. 1991.《Core Wars Genetics：捕食的进化》。（1991）。https://corewar.co.uk/perry/evolution.htm [61] Ulyana Piterbarg, Lerrel Pinto, and Rob Fergus. 2023. 神经语言智能体的差异补丁历史。arXiv 预印本 arXiv:2312.07540 (2023)。[62] Rémy Portelas, Cédric Colas, Lilian Weng, Katja Hofmann, and Pierre-Yves Oudeyer. 2020. 深度强化学习的自动课程学习：简要综述。arXiv 预印本 arXiv:2003.04664 (2020)。[63] Steen Rasmussen, Carsten Knudsen, Rasmus Feldberg, and Morten Hindsholm. 1990. Coreworld：计算化学中的合作结构涌现与进化。《物理学 D：非线性现象》42, 1-3 (1990), 111–134。[64] Thomas S. Ray. 1991. 数字生物的进化与优化。《超级计算中的科学卓越》（1991）。[65] Esteban Real, Chen Liang, David So, and Quoc Le. 2020. AutoML-Zero：从零开始进化机器学习算法。载于《国际机器学习会议》。PMLR，8007–8019。[66] Matt Ridley. 1994.《红皇后：性与人性的进化》。Penguin UK。[67] Sebastian Risi, Yujin Tang, David Ha, and Risto Miikkulainen. 2025. 神经进化：运用创造力进行人工智能智能体设计。麻省理工学院出版社，剑桥，马萨诸塞州。https://neuroevolutionbook.com [68] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. 2024. 通过大语言模型进行程序搜索获得的数学发现。《自然》625, 7995 (2024), 468–477。[69] C. D. Rosin and R. K. Belew. 1997. 竞争性协同进化的新方法。《进化计算》 5, 1 1997 1 29 10 1162 1997 5 1 1 ( ), – 。doi: 。/ [70] 1959 evco. . . . Arthur L Samuel. . 使用西洋跳棋进行机器学习的一些研究。《IBM 研究与开发杂志》3, 3 (1959), 210–229。[71] Mikayel Samvelyan, Sharath Chandra Raparthy, Andrei Lupu, Eric Ham- bro, Aram H. Markosyan, Manish Bhatt, Yuning Mao, Minqi Jiang, Jack Parker-Holder, Jakob Foerster, Tim Rocktäschel, and Roberta Raileanu. 2024. Rainbow Teaming：开放式生成多样化的对抗提示。arXiv: 2402 16822 [ ] 2402 16822 . cs.CL [72] https://arxiv.org/abs/ 。Ernesto Sanchez, Massimiliano Schillaci, and Giovanni Squillero. 2006. 为 Nano Core 进化战士程序。载于《2006 年 IEEE 计算智能与博弈研讨会》。IEEE，272–278。[73] Joseph A Schumpeter. 1942.《资本主义、社会主义与民主》。Routledge。[74] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion：具备语言强化学习能力的语言智能体。《神经信息处理系统大会进展》36 (2023), 8634–8652。[75] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershel- vam, Marc Lanctot, et al. 2016. 通过深度神经网络和树搜索掌握围棋。《自然》529, 7587 (2016), 484–489。[76] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2017. 通过通用强化学习算法进行自博弈，掌握国际象棋与将棋。arXiv 预印本 arXiv:1712.01815 (2017)。[77] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017 550, 7676 2017 354 359. 无需人类知识掌握围棋。《自然》（），–。[78] Joel Simon. 2025. 使用推理大语言模型进行创造性探索（Lluminate）。https: // www.joelsimon.net/lluminate. https://www.joelsimon.net/lluminate [79] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. 最优地扩展大语言模型测试时计算，可能比扩展模型参数更有效。arXiv 预印本 arXiv:2408.03314 (2024)。[80] Kenneth O Stanley and Joel Lehman. 2015.《为何伟大无法规划：目标的神话》。Springer。[81] Kenneth O Stanley and Risto Miikkulainen. 2004. 通过进化复杂化实现竞争性协同进化。《人工智能研究杂志》21 (2004), 63–100。[82] Yujin Tang, Jie Tan, and Tatsuya Harada. 2020. 通过对抗训练学习敏捷运动。载于《2020 IEEE/RSJ 智能机器人与系统国际会议》（IROS）。IEEE，6098–6105。[83] Humayra Tasnim and William Collishaw. 2016. 使用遗传算法进化 Core War 战士。（2016）。

[84] Gerald Tesauro等人。1995年。时序差异学习与TD-Gammon。《*ACM*通讯》38卷第3期（1995年），58–68页。[85] Leigh Van Valen。1973年。一条新的进化定律。《进化理论》1（1973年），1–30页。[86] Oriol Vinyals、Igor Babuschkin、Wojciech M Czarnecki、Michaël Mathieu、An- drew Dudzik、Junyoung Chung、David H Choi、Richard Powell、Timo Ewalds、Petko Georgiev等人。2019年。使用多智能体强化学习达到《星际争霸II》宗师级水平。《自然》575卷，7782期（2019年），350–354页。[87] Haiyang Wang、Zhiliang Tian、Yuchen Pan、Xin Song、Xin Niu、Minlie Huang和Bin Zhou。2025年。迎战强硬抵抗者：通过对抗性博弈为非协作对话制定策略规划。载于《计算语言学协会第*63*届年会论文集（第*1*卷：长篇论文）》。3665–3685页。[88] Rui Wang、Joel Lehman、Jeff Clune和Kenneth O. Stanley。2019年。成对开放式拓荒者（POET）：持续生成日益复杂、多样的学习环境及其解决方案。arXiv:1901.01753 [cs.NE] https://arxiv.org/abs/ 1901 01753 。Rui Wang、Joel Lehman、Aditya Rawal、[89] Jiale Zhi、Yulun Li、Jeff Clune和Kenneth O. Stanley。2020年。增强版POET：通过无限发明学习挑战及其解决方案实现开放式强化学习。arXiv: 2003 08536 [ ] 2003 08536 。cs.NE [90] 2001 https://arxiv.org/abs/ 。Richard A Watson和Jordan B Pollack。。最小基底中的协同进化动态。载于《遗传与进化计算会议论文集（*GECCO 2001*）》。702–709页。[91] Yuxiang Wei、Zhiqing Sun、Emily McMilin、Jonas Gehring、David Zhang、Gabriel Synnaeve、Daniel Fried、Lingming Zhang和Sida Wang。2025年。通过自博弈式SWE-RL训练超级智能软件智能体。*arXiv*预印本 *arXiv:2512.18552*（2025年）。[92] Stephen Wolfram和M Gad-el Hak。2003年。一种新型科学。《应用力学评论》56卷第2期（2003年），B18–B19页。[93] Larry Yaeger。1994年。计算遗传学、生理学、代谢、神经系统、学习、视觉与行为，或PolyWorld：新语境中的生命。载于《圣菲研究所复杂性科学研究丛书：人工生命*III*会议论文集》，第17卷。Addison-Wesley，263–263页。[94] Huining Yuan、Zelai Xu、Zheyue Tan、Xiangmin Yi、Mo Guang、Kaiwen Long、Haojia Hui、Boxun Li、Xinlei Chen、Bo Zhao等人。2025年。MARSHAL：通过与战略型大语言模型进行自博弈，激励多智能体推理。*arXiv*预印本 *arXiv:2510.15414*（2025年）。[95] Jie Zhang、Haoyu Bu、Hui Wen、Yongji Liu、Haiqiang Fei、Rongrong Xi、Lun Li、Yun Yang、Hongsong Zhu和Dan Meng。2025年。当大语言模型遇上网络安全：系统性文献综述。《网络安全》8卷第1期（2025年），55。[96] Jenny Zhang、Shengran Hu、Cong Lu、Robert Lange和Jeff Clune。2025年。达尔文-哥德尔机器：自我改进智能体的开放式进化。*arXiv*预印本 *arXiv:2505.22954*（2025年）。[97] Jenny Zhang、Joel Lehman、Kenneth Stanley和Jeff Clune。2023年。Omni：通过人类对趣味性的认知模型实现开放性。*arXiv*预印本 *arXiv:2306.01711*（2023年）。[98] Ruize Zhang、Zelai Xu、Chengdong Ma、Chao Yu、Wei-Wei Tu、Wenhao Tang、Shiyu Huang、Deheng Ye、Wenbo Ding、Yaodong Yang等人。2024年。强化学习中的自博弈方法综述。*arXiv*预印本 *arXiv:2408.01072*（2024年）。

A Core War 详述 A.1 Redcode 详述

Redcode 操作码。 Redcode 程序由一小组类汇编指令组成。操作码包括：

- **进程控制：** DAT 终止当前进程；SPL 在目标地址生成一个新进程；NOP 不执行任何操作；ORG 指定程序入口点；END 标记程序的结束。
- **数据移动与算术：** MOV 复制数据或指令；ADD、SUB、MUL、DIV 和 MOD 对指令字段执行算术运算，并将结果写入内存（除法／取模遇到零除数时会终止该进程）。
- **控制流：** JMP 执行无条件跳转；JMZ 和 JMN 根据零值/非零值测试执行有条件跳转；DJN 将值递减并执行有条件跳转。

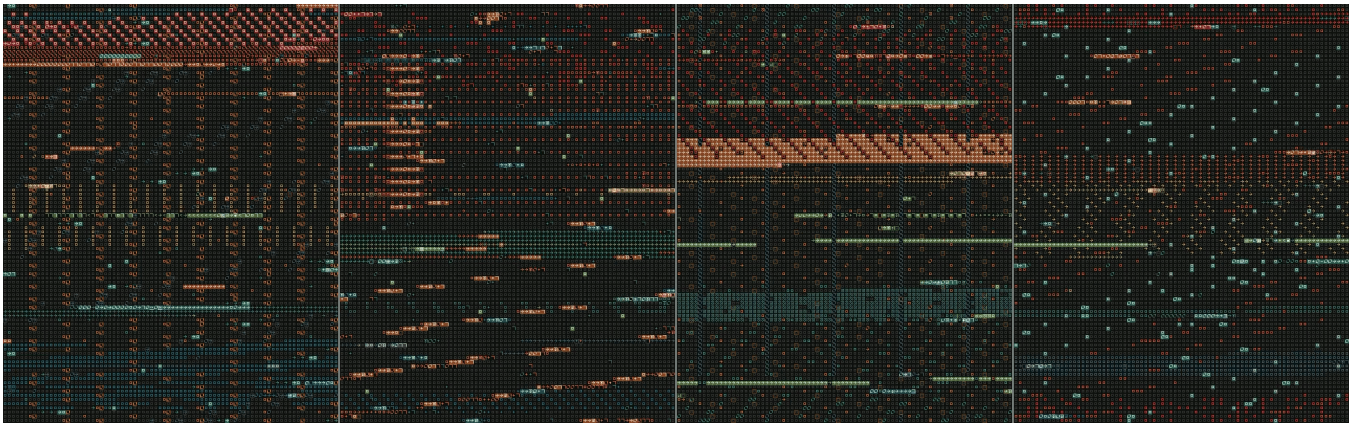


图10：DRQ 发现的战士之间相互竞争的可视化结果。尽管整体功能在统计上趋于收敛，该图仍凸显了通过自博弈产生的策略多样性。

• **比较与分支：** SEQ/CMP 在操作数相等时跳过下一条指令；SNE 在操作数不相等时跳过；SLT 在一个操作数小于另一个操作数时跳过。• **汇编指令：** EQU 定义符号常量。

操作码可以添加修饰符和寻址模式，以决定读取或写入哪些指令字段，以及如何计算内存地址。

Redcode 指令修饰符。Redcode 操作码后面可以加上点号和修饰符，以指定其操作的字段：

• **.A**— 对 A 数值进行操作并写入 A 数值。• **.B**— 对 B 数值进行操作并写入 B 数值。

• **.AB** — 使用 A 字段的 A 数值和 B 字段的 B 数值；写入 B 数值。

• **.BA** — 使用 A 字段的 B 数值和 B 字段的 A 数值；写入 A 数值。

• **.F** — 并行使用 A 数值和 B 数值；分别写入两者（A 到 A，B 到 B）。• **.X** — 跨字段操作；同时写入两者（A 到 B，B 到 A）。• **.I** — 对完整指令进行操作并写入完整指令。

Redcode 寻址模式。Redcode 指令支持多种操作数寻址模式：

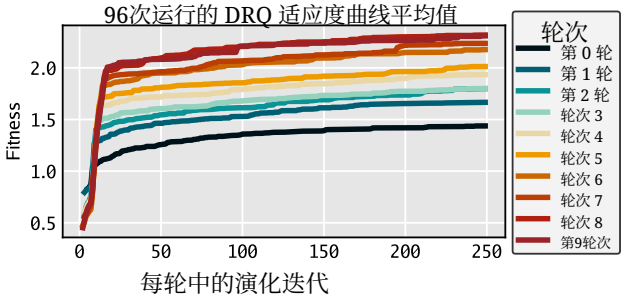


图 11：所有 DRQ运行的平均 DRQ 适应度曲线。

• **立即寻址#：** 操作数是字面数据；将 A/B 指针设为零。

• **直接寻址 $：** 操作数是相对于程序计数器的偏移量；A/B 指针复制当前指令的 A/B 数值。

• **A 数间接寻址*：** 通过另一条指令的 A 字段，利用主偏移量定位次级偏移量；A/B 指针是当前指令的 A/B 数与被引用指令的 A 数之和。

• **B 数间接寻址 @：** 与 A 数间接寻址类似，但使用被引用指令的 B 字段。

• **A 数前置递减间接寻址 {：** 与 A 数间接寻址类似，但在使用前先递减被引用指令的 A 字段。

• **B 数前置递减间接寻址 <：** 与 B 数间接寻址类似，但在使用前先递减 B 字段。

• **A 数后置递增间接寻址 }：** 与 A 数间接寻址类似，但在操作数求值后递增被引用指令的 A 字段。

• **B 数后置递增间接寻址 >：**与 B 数间接寻址类似，但在求值后递增被引用指令的 B 字段。

A.2 CoreWar 模拟详情

Core War 详情。 评估战士程序之间的对战时，结果取 20 次相互独立模拟的平均值，战士初始位置随机化。所有实验均使用包含 8,000 个地址的 Core，并运行最多 80,000 个模拟时间步。每个战士程序最多可生成 8,000 个并发线程。战士程序的源代码长度限制为 100 条指令，并进行初始化，使不同战士程序的起始位置至少相隔 100 条指令。

Core War 代码库。 我们使用以下 Python Core War 实现和渲染器：
https://github.com/rodrigosetti/corewar

我们封装这段代码，以处理大语言模型利用的边缘情况（例如指数增长会产生天文数字般巨大的整数），并

监控战士属性，例如生成的线程总数和内存覆盖率。我们的修改已在代码仓库中提供。

战士评估。 评估战士的通用性时，我们会测量它们与从以下代码仓库中抓取的317个人类战士的对战表现：

- https://github.com/rodrigosetti/corewar • https://github.com/n1LS/redcode-warriors

B 大语言模型提示词

我们向大语言模型提供了一个固定的系统提示词，要求它担任 Core War 编程游戏的编程助手。该提示词包含一份自包含的 Core War 环境规范，其中包括对 Redcode 的描述、完整的指令集（操作码、修饰符和寻址模式）、执行语义以及语法规则。提示词中还包含若干经典的 Redcode 程序，作为说明性示例。最后，提示词对程序结构施加了严格限制（例如，必须使用 ORG 起始指令和 END 指令、遵循标签使用规则以及采用相对寻址），确保生成的所有战士程序在语法上有效，并可在 Core War 模拟器中执行。

确切的提示词可在我们的代码仓库中找到。

C 附加图表

本节展示我们大规模 DRQ 实验的更多图表。

图12展示了所有 DRQ运行的适应度曲线。图11展示了平均适应度曲线。图13展示了随机抽取的若干轮次对应的 MAP-Elites 网格。图10展示了发现的战士程序在 Core War 模拟中相互对战的示例。



图12：全部96次DRQ运行的适应度曲线。为便于观察，我们仅展示每次运行前10轮的适应度曲线。

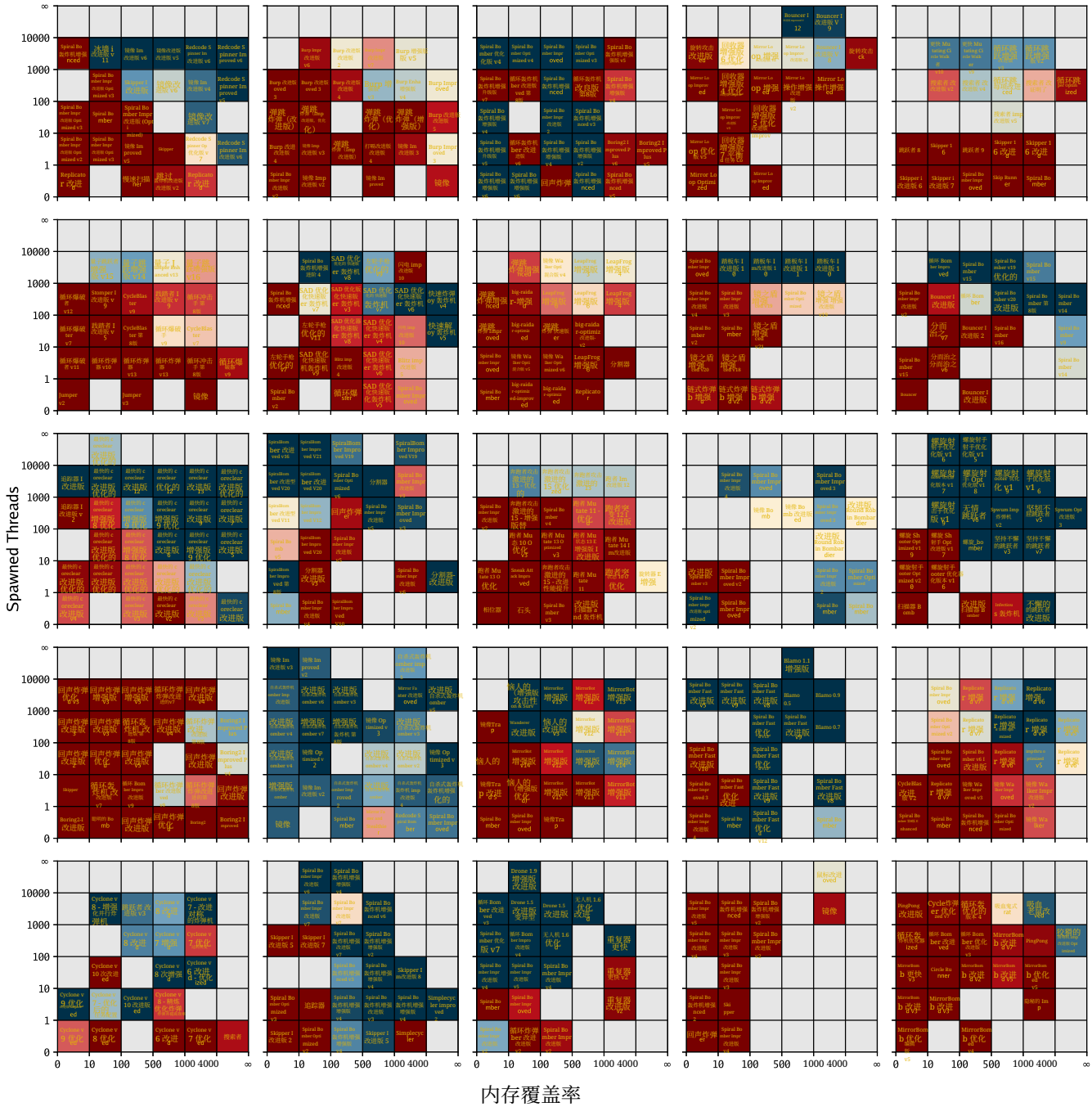


图 13：从随机选取的轮次和运行中得到的 DRQ MAP-Elites档案示例。每个分箱根据精英战士的适应度着色（红色 = 较差，蓝色 = 较好），并标注精英战士的名称。